The Complexity of Querying External Memory and Streaming Data

Martin Grohe[1], Christoph Koch[2], and Nicole Schweikardt[1]

[1] Institut für Informatik, Humboldt-Universität Berlin, Germany
{grohe,schweika}@informatik.hu-berlin.de
[2] Database Group, Universität des Saarlandes, Saarbrücken, Germany
koch@cs.uni-sb.de

Abstract. We review a recently introduced computation model for streaming and external memory data. An important feature of this model is that it distinguishes between *sequentially reading (streaming)* data from external memory (through main memory) and *randomly accessing* external memory data at specific memory locations; it is well-known that the latter is much more expensive in practice. We explain how a number of lower bound results are obtained in this model and how they can be applied for proving lower bounds for XML query processing.

1 Introduction

Modern computers rely on a hierarchy of storage media from tapes and disks at the bottom through what is usually called random-access memory or main memory up to various levels of (even on-CPU) memory caches at the top. The storage media at the bottom of this hierarchy are the slowest and least expensive and those at the top the fastest and dearest. The need for this *memory hierarchy* is dictated by the ever-growing amounts of data that have to be managed and processed by computers. Currently, the most pronounced performance and price gap in this hierarchy is between (random access) main memory and the next-lower level in the memory hierarchy, usually magnetic disks, which have to rely on mechanical, physically moving parts. One often refers to the upper layers above this gap by *internal memory* and the lower layers of the memory hierarchy by *external memory*. The technological reality is such that the time for accessing a given bit of information in external memory is five to six orders of magnitude greater than the time required to access a bit in internal memory.

Current external storage technology (disks and tapes) renders algorithms that can read and write their data to and from external memory in few *sequential scans* much faster than algorithms that require many random data accesses. Indeed, the time required to move a read/write head to a certain position of a disk or tape – a slow mechanical operation – is by orders of magnitude greater than actually reading a considerable amount of data stored in sequence once the read/write head has been placed at the starting position of the data in question.

Managing and processing huge amounts of data has been traditionally the domain of database research. It is generally assumed that databases have to reside in external, inexpensive storage because of their sheer size. There has been a wealth of research on query processing and optimization respecting the mentioned physical realities and distinguishing between internal and external memory (cf. e.g. [20, 10, 24, 16]). In fact, this distinction is in a sense the defining essence of database techniques.

The fundamental problems that have to be faced in processing very large datasets have generated a recent renewed interest in theoretical aspects of external memory processing. In the classical model of external memory algorithms (see, for example, [24, 16]), the cost measure is simply the number of bits read from external memory divided by the page size. This model ignores the very important distinction mentioned above between *random access* to data in particular external memory locations and *sequential scans* of the disks. More recent models focus on data processing with few sequential scans of the external memory [2, 14, 4, 12]. An important special case is *data stream processing*, in which only one sequential scan of the data is permitted. This is yet another field that has seen much activity in recent years [18, 1, 5].

In [11, 12], we introduced a formal model for external memory processing that allows to distinguish between sequential scans and random access. The two most significant cost measures in our setting are the number of random accesses to external memory and the size of the internal memory. Our model is based on standard multi-tape Turing machines. Our machines have several tapes of unbounded size, among them the input and output tapes, which represent the external memory (for example, each of these tapes may represent a disk). In addition, the machine has several tapes of restricted size which represent the internal memory.

We model the number of scans of the external memory data, respectively the number of random accesses, by the number of reversals of the Turing machine's read/write heads on the external memory tapes. Anything close to random I/O will result in a very considerable number of reversals, while a full sequential scan of an external memory tape can be effected cheaply. The reversals done by a read/write head are a clean and fundamental notion [25], but of course real external storage technology based on disks does not allow to reverse their direction of rotation. On the other hand, we can of course simulate k forward scans by $2k$ reversals in our machine model — and allowing for forward as well as backward scans makes the *lower* bound results presented in this paper even stronger.

Note that our model puts no restriction on the number of head reversals on the *internal* memory tapes, the size of the external memory tapes, or the running time of the machine.

In this paper, we give a survey of the above-mentioned machine model and strong lower bounds that were recently obtained for it [11, 12, 13]. We start in Section 2 by formally introducing the machine model and showing a number of basic properties for it. In Section 3, we consider the case of machines with only a single external memory tape. Here we can employ techniques from communication complexity to obtain lower bounds. In Section 4, we apply the results of

Section 3 to XML query processing problems. XML is a data exchange format that is currently drawing much attention in data management research. In [11], we obtained lower bounds for processing queries in the languages XQuery and XPath, the two most widely used query languages for XML data (in fact, XPath is basically a sublanguage of XQuery that is also often used in isolation and has become part of other XML-related data transformation languages such as XSLT). Section 5 goes beyond the case of machines with a single external memory tape. We will see that techniques from communication complexity fail to prove lower bounds. In [12], we introduced a new technique to establish lower bounds in this model. These are based on a new (non-uniform) machine model, so-called *list machines*, which allow us to analyze the flow of information in a Turing machine computation. The main result is a lower bound for the sorting problem.

Related Work

Most results presented in this survey are due to [11, 12, 13].

Strong lower bounds for a number of problems are known in models which permit a small number of sequential scans of the input data, but no auxiliary external memory (that is, the version of our model with no external memory tapes besides the input tape) [1, 2, 3, 4, 5, 6, 14, 17, 18]. All these lower bounds are obtained by communication complexity.

In [3], the problem of determining whether a given relational query can be evaluated scalably on a data stream or not at all is addressed. The complexity of XML query evaluation in a streaming model is also addressed in [5, 6]. The time and space complexity of XPath query evaluation in the standard (main memory) model is studied in [8, 9, 23].

Obviously, our model is also related to the *bounded reversal Turing machines*, which have been studied in classical complexity theory (see, for example, [25]). However, in bounded reversal Turing machines, the number of head reversals is limited on *all* tapes, whereas in our model there is no such restriction on the internal memory tapes. This makes our model considerably stronger. In particular, in our lower bound results we allow internal memory size that is polynomially related to the input size.

2 The Machine Model

Our model is based on standard multitape Turing machines. If not explicitly mentioned otherwise, we assume our machines to be deterministic. The machines we consider have $t + u$ tapes. The first t tapes are called *external memory tapes*; they represent external memory devices such as hard disks. The other u tapes are called *internal memory tapes*; they represent the internal memory. The first tape is always viewed as the input tape. If necessary, the machines have an additional write-only output tape. Configurations, runs, and acceptance are defined in the usual way. Figure 1 illustrates our model.

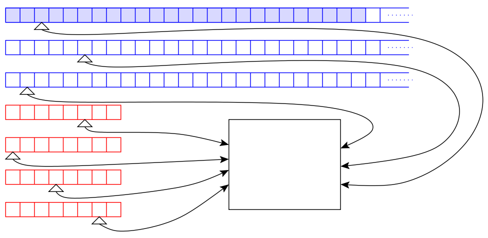

Fig. 1. Our machine model

Let M be such a Turing machine and ρ a run of M. The *(internal) space* required by ρ, $\mathrm{space}(\rho)$, is the total number of cells on the internal memory tapes visited during the run ρ. For $1 \leq i \leq t$, the number of *head reversals* on the i-th external memory tape, $\mathrm{rev}(\rho, i)$, is the number of times the read/write head on tape i changes its direction during the run ρ.

For functions $r, s : \mathbb{N} \to \mathbb{N}$ (where $\mathbb{N}$ denote the set of positive integers), we call the machine M (r, s, t)-*bounded* if it has t external memory tapes and for every run ρ of M with an input of length N and $i \in \{1, .., t\}$ we have $1 + \sum_{i=1}^{t} \mathrm{rev}(\rho, i) \leq r(N)$ and $\mathrm{space}(\rho) \leq s(N)$.

Definition 1. *(1) For functions $r, s : \mathbb{N} \to \mathbb{N}$ and $t \in \mathbb{N}$ we let $\mathrm{ST}(r, s, t)$ be the class of all problems that can be decided by an (r, s, t)-bounded Turing machine.*

(2) For classes R, S of functions we let $\mathrm{ST}(R, S, t) = \bigcup_{\substack{r \in R \\ s \in S}} \mathrm{ST}(r, s, t)$. Furthermore, we let $\mathrm{ST}(R, S, O(1)) = \bigcup_{t \in \mathbb{N}} \mathrm{ST}(R, S, t)$.

(3) We let $\mathrm{ST}(R, S) = \mathrm{ST}(R, S, 1)$.

While usually by "problem" we mean "decision problem", that is, language over some finite alphabet, occasionally we are more liberal and also view partial functions as problems. In particular, we may write $f \in \mathrm{ST}(r, s, t)$ for a partial function $f : \Sigma^* \to \Sigma^*$.

Occasionally, we also consider the nondeterministic versions $\mathrm{NST}(\dots)$ of our $\mathrm{ST}(\dots)$ classes.

Note that we do not restrict the running time of an (r, s, t)-bounded machine in any way. Neither do we restrict the external space, that is, the number of cells on the external memory tapes that are visited during a run. However, it is not hard to see that implicitly the running time and hence the external space are bounded in terms of reversals and internal space:

Lemma 2 ([12]). *Let $r, s : \mathbb{N} \to \mathbb{N}$ and $t \in \mathbb{N}$, and let M be an (r, s, t)-bounded Turing machine. Then the length of every finite run of M is at most*

$$N \cdot 2^{O(r(N) \cdot (t + s(N)))}.$$

For $(r, s, 1)$-bounded Turing machines, the bound can be improved to

$$(N + r(N)) \cdot r(N) \cdot 2^{O(s(N))},$$

as an easy induction on $r(N)$ shows.

Random access

If we think of the first t tapes of an (r, s, t)-bounded Turing machine as representing hard disks, then admitting heads to reverse their direction may not be very realistic. But as we mainly use our model to prove *lower* bounds, it does not do any harm either. Head reversals are a convenient way to simulate *random access* in our model.

Alternatively, we can explicitly include random access into our model as follows: A *random access Turing machine* is a Turing machine which has a special *address tape* on which only binary strings can be written. These binary strings are interpreted as nonnegative integers specifying external memory addresses, that is, numbers of cells on the external memory tapes. For each external memory tape $i \in \{1, \ldots, t\}$ the machine has a special state ra_i. If ra_i is entered, then in one step the head on tape i is moved to the cell that is specified by the number on the address tape, and the content of the address tape is deleted. Figure 2 illustrates the augmented model.

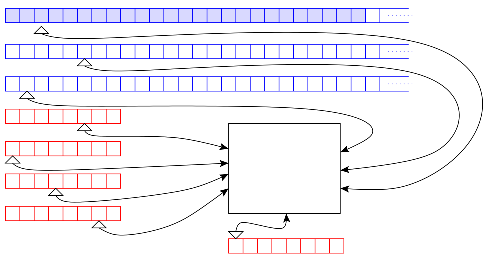

Fig. 2. Random access Turing machine

Let $q, r, s : \mathbb{N} \to \mathbb{N}$ and $t \in \mathbb{N}$. A random access Turing machine T is (q, r, s, t)-*bounded* if it is (r, s, t)-bounded (in the sense of a standard Turing machine) and, in addition, every run ρ of T on an input of length N involves at most $q(N)$ random accesses.

The address tape is considered as part of the internal memory; thus in a (q, r, s, t)-bounded random access Turing machine the length of the address tape is bounded by $s(N)$, where N is the length of the input. This implies that we can only address the first $2^{s(N)}$ cells of the external memory tapes. If during a computation, these tapes get longer, we only have random access to initial segments of length $2^{s(N)}$.

The following lemma follows from the simple observation that each random access can be simulated by moving the head to the desired position. This simulation is possible with at most two head reversals (if we want the head to be headed in the same direction before and after the simulation of the random access) and a slight space overhead.

Lemma 3 ([12]). *Let $q, r, s : \mathbb{N} \to \mathbb{N}$ and $t \in \mathbb{N}$. Then if a problem can be solved by a (q, r, s, t)-bounded random access Turing machine, it can also be solved by an $(r + 2q, O(s), t)$-bounded Turing machine.*

From now on, we will focus on standard Turing machines without address tapes.

Let us summarize the classes we have defined. For functions r, s on the natural numbers:

- $\mathrm{ST}(O(r), O(s), O(1))$ is the class of all problems that can be solved on a machine with internal memory size $O(s(N))$ and an arbitrary (albeit constant) number of arbitrarily large external memory devices, which can be read sequentially, in addition allowing at most $O(r(N))$ head reversals and random accesses. The first external memory device contains the input data.
- $\mathrm{ST}(O(r), O(s))$ is the class of all problems that can be solved on a machine that only has one external memory device and is otherwise restricted as above. Essentially, $\mathrm{ST}(O(r), O(s))$ is the class of all problems that can be solved on an $O(s(N))$-space bounded machine allowing at most $O(r(N))$ sequential scans of the input.[3]
- $\mathrm{ST}(1, O(s))$ is the class of all problems that can be solved on an $O(s(N))$-space bounded machine if the input is a data stream.

We always use N to denote the input size.

3 Lower Bounds via Communication Complexity

Lower bounds for the $\mathrm{ST}(r, s)$-model can be obtained fairly easily by employing known results from communication complexity. The idea is to divide the input tape into two parts, which induces a split of the input data into two parts. Now we ask how much information must be communicated between the two parts to answer a specific query. Suppose we can prove a lower bound of $c(N)$ for the number of bits that need to be communicated, where N denotes the size of the input. Then we also have a lower bound $r(N) \cdot s(N) \geq \Omega(c(N))$ for functions r, s such that the query can be answered in $\mathrm{ST}(r, s)$, because each time we cross the dividing line between the two parts of the input, we can only "transport" $O(s(N))$ bits of information.

In the basic model of communication complexity [26], two players, Alice and Bob, jointly want to evaluate $F(x, y)$, where $F : A \times B \to C$ is a function defined on finite sets A, B. Alice is given the first argument $x \in A$ and Bob the second argument $y \in B$. The two players exchange messages according to some fixed protocol until one of them has gathered enough information to compute $F(x, y)$. The *cost* of the protocol is the maximum number of bits communicated, where the maximum is taken over all argument pairs $(x, y) \in A \times B$. The *communication complexity of* F is the minimum of the costs of all protocols computing F.

[3] $\mathrm{ST}(O(r), O(s))$ is slightly more powerful because the input can be overwritten and the external memory device can be used for storing auxiliary data.

A function that almost obviously has a high communication complexity is the *disjointness function* $\textsc{disj}_\ell$, defined on subsets $x, y \subseteq \{0, \ldots, \ell - 1\}$ by

$$\textsc{disj}_\ell(x, y) = \begin{cases} 1 & \text{if } x \cap y = \emptyset, \\ 0 & \text{otherwise.} \end{cases}$$

Indeed, it is easy to see that the communication complexity of $\textsc{disj}_\ell$ is ℓ. It will be convenient for us to work with a slight modification of the disjointness function. For $k \leq \ell$, let $\textsc{disj}_{\ell,k}$ be the restriction of $\textsc{disj}_\ell$ to pairs of k-element subsets of $\{0, \ldots, \ell - 1\}$.

Theorem 4 ([21], cf. Example 2.12 in [15]). *The communication complexity of* $\textsc{disj}_{\ell,k}$ *is* $\Omega\left(\log \binom{\ell}{k}\right)$.

Let us now consider the following decision problem:

Disjoint-Sets
Instance: Strings $x_1, \ldots, x_m, y_1, \ldots, y_m \in \{0,1\}^n$.
Question: Is $\{x_1, \ldots, x_m\} \cap \{y_1, \ldots, y_m\} = \emptyset$?

Formally, the input may either be specified as the string

$$x_1 \# x_2 \# \ldots \# x_m \# y_1 \# \ldots \# y_m$$

over $\{0, 1, \#\}$ or as an XML-document as described in Section 4 (see Figure 3). In both cases, the input size N is $\Theta(m \cdot n)$.

The lower bound of the following corollary follows easily from Theorem 4. To see this, we consider instances with $n = 2 \log m$ and note that for such instances we have $N = \Theta(m \cdot \log m)$ and, with $\ell = 2^n$ and $k = m$,

$$\log \binom{\ell}{k} = \log \binom{m^2}{m} \geq \log \left(m^m\right) = m \cdot \log m.$$

The upper bound of the corollary is trivial.

Corollary 5. *(1)* Disjoint-Sets $\in \mathrm{ST}(1, N)$.
(2) For all functions $r, s : \mathbb{N} \to \mathbb{N}$ *with* $r(N) \cdot s(N) \in o(N)$,

$$\textsc{Disjoint-Sets} \notin \mathrm{ST}(r, s).$$

In the next section, we will apply this simple result to prove lower bounds for querying XML-documents.

Next, we separate the deterministic from the nondeterministic ST-classes. We observe that the complement of Disjoint-Sets can be decided by a $(1, n, 1)$-bounded *nondeterministic* Turing machine by guessing an input string x_i, storing it in the internal memory, and comparing with all strings y_j. Thus the restriction of the complement of Disjoint-Sets to inputs with $n = 2 \cdot \log m$ is in $\mathrm{NST}(1, O(\log N))$. The proof of Corollary 5 shows that it is not in $\mathrm{ST}(r, s)$ for all functions r, s with $r(N) \cdot s(N) \in o(N)$. Thus:

Corollary 6. *For all functions r, s with $r(N) \cdot s(N) \in o(N)$*

$$\mathrm{NST}(1, O(\log N)) \not\subseteq \mathrm{ST}(r, s).$$

The following hierarchy theorem is based on a more sophisticated result from communication complexity that deals with the number (and not only size) of messages Alice has to send Bob in a communication protocol. For functions $s : \mathbb{N} \to \mathbb{N}$ and $k \in \mathbb{N}$, let $\mathrm{ST}(k, s)$ denote the class $\mathrm{ST}(r, s)$ with $r(N) = k$ for all $N \in \mathbb{N}$.

Theorem 7 ([11], based on [7]). *For every fixed $k \in \mathbb{N}$ and all classes S of functions from $\mathbb{N}$ to $\mathbb{N}$ such that $O(\log n) \subseteq S \subseteq o\big(\frac{\sqrt{N}}{(\lg n)^3}\big)$ we have*

$$\mathrm{ST}(k, S) \subsetneq \mathrm{ST}(k{+}1, S).$$

4 Applications to XML Query Processing

In this section, we show how the results and techniques of the previous section can be applied to prove lower bounds for the complexity of XML-query processing. We assume that the reader is vaguely familiar with XML-syntax. We will only use very basic XML consisting of opening tags `<t>` and the corresponding closing tags `</t>` and plain text (no attributes, no DTDs, et cetera).

As an example, let us encode instances of the DISJOINT-SETS problem in XML. An instance $x_1, \ldots, x_m, y_1, \ldots, y_m \in \{0,1\}^n$ is represented by the XML-document displayed in Figure 3.

```
<instance>
  <set1>
    <string> x_1 </string> ... <string> x_m </string>
  </set1>
  <set2>
    <string> y_1 </string> ... <string> y_m </string>
  </set2>
</instance>
```

Fig. 3. XML-representation of the two m-element sets $\{x_1, \ldots, x_m\}$ and $\{y_1, \ldots, y_m\}$

We will talk about the XML query languages XQuery and XPath, but the reader is not expected to know these languages. To avoid the awkward term "XQuery query", we refer to queries in the language XQuery as *xqueries*. An xquery Q transforms a given document D into a document $Q(D)$.

As an example, consider the xquery displayed in Figure 4. The syntax is, to a certain extent, self-explanatory. The query transforms the XML-document in Figure 3 to the document in Figure 5, where $1 \leq i_1 < i_2 < \cdots < i_\ell \leq m$ such

```
<result>
  for $x in /instance/set1/string
  where some $y in /instance/set2/string satisfies $x = $y
  return $x
</result>
```

Fig. 4. An xquery computing the intersection of two sets

```
<result>
  <string> $x_{i_1}$ </string> ... <string> $x_{i_\ell}$ </string>
</result>
```

Fig. 5. Result of applying the query in Figure 4 to the document in Figure 3

that $\{x_1, \dots, x_m\} \cap \{y_1, \dots, y_m\} = \{x_{i_1}, \dots, x_{i_\ell}\}$. Thus the query computes the intersection of the two sets. By the results of the previous section, the query cannot be evaluated by an $(r, s, 1)$-bounded Turing machine for any functions r, s with $r(N) \cdot s(N) \in o(N)$.

We can associate the following decision problem with the evaluation problem for a query Q:

Q-Filtering
Instance: XML-document D.
Question: Is $Q(D)$ nonempty ?

Here we call an XML-document *empty* if it just consists of an opening and closing tag, as for example `<result> </result>`. We ignore whitespaces.

The Q-Filtering problem for the query Q of Figure 4 is the complement of Disjoint-Sets. Thus we get:

Corollary 8. *There is an xquery Q such that for all functions r, s with $r(N) \cdot s(N) \in o(N)$,*

$$Q\text{-Filtering} \notin \mathrm{ST}(r, s).$$

XPath is a *node selecting language*; the result of applying an XPath query to an XML-document is a set of nodes of this document. A *node* of a document is pair of corresponding opening- and closing tags. The nodes are arranged in a tree-like fashion in the obvious way. Thus we can view XML-documents as labeled rooted trees. Inner nodes are labeled by tags and leaves by the text parts of the document. Node selection in XPath works by regular expressions over the tags which specify paths in a document (tree). There is no need for the reader to know any details about the language here. XPath is mainly used as a tool in more complicated XML-related formalisms such as XQuery or XML-Schema. As far as expressive power is concerned, XPath is strictly a fragment of XQuery. For our complexity lower bounds, we only consider a small fragment of XPath called Core-XPath [8]. It is a clean logical fragment that captures the core functionality of the much larger and messier language XPath. We define Q-FILTERING for XPath queries Q as for xqueries. Here, $Q(D)$ is a set of nodes of D and emptiness has the natural meaning.

To measure the complexity of Q-FILTERING for XPath queries Q, the appropriate parameter is not the size of the input document but its *height* if viewed as a tree. We use the notation $\mathrm{ST}(r, s)$ for functions r, s depending on the height and not the size of the input document with the obvious meaning.

Theorem 9 ([11]).

(1) For every Core-XPath query Q,

$$Q\text{-FILTERING} \in \mathrm{ST}(1, O(h)).$$

(2) There is a Core-XPath query Q such that for all functions r, s with $r(h) \cdot s(h) \in o(h)$,

$$Q\text{-FILTERING} \notin \mathrm{ST}(r(h), s(h)).$$

Here h denote the height of the input document.

The upper bound is proved by standard automata theoretic techniques (implicitly, the result can be found in [19, 22]). For the lower bound, we again use the DISJOINT-SETS problem, but encoded as an XML-document in a different way than before.

5 Lower Bounds via List Machines

In this section, we turn to the classes $\mathrm{ST}(r, s, t)$ for $t > 1$. To prove lower bounds for these classes, communication complexity based arguments as used in Section 3 utterly fail. The reason is that we can easily communicate arbitrarily many bits from one part of the input to any other part just by copying the first part to a second external memory tape and then reading it in parallel with the second part. This requires no internal memory and just two head reversals.

This idea can be used to prove that the $\mathrm{ST}(r, s, 2)$ classes are much more powerful than the $\mathrm{ST}(r, s) = \mathrm{ST}(r, s, 1)$ classes. Observe that Theorem 4 not only yields a lower bound for the DISJOINT-SETS problem, but also for its restriction to input sets which are ordered in any specific way, because the communication complexity lower bound of Theorem 4 is independent of the way the two arguments are given to the function $\mathrm{DISJ}_{k,\ell}$. Moreover, we obtained the lower bound of Corollary 5 for instances with $n = 2 \cdot \log m$. Thus for all functions $r, s : \mathbb{N} \to \mathbb{N}$ with $r(N) \cdot s(N) \in o(N)$ the following problem is not in $\mathrm{ST}(r, s)$.

Instance:	Strings $x_1, \ldots, x_m, y_1, \ldots, y_m \in \{0,1\}^{2\cdot\log m}$ such that $x_m \leq x_{m-1} \leq \cdots \leq x_1$ and $y_1 \leq y_2 \leq \ldots \leq y_m$.
Question:	Is $\{x_1, \ldots, x_m\} \cap \{y_1, \ldots, y_m\} = \emptyset$?

Here $\leq$ denotes the lexicographical order of strings over $\{0,1\}$.

By copying all x_i in the order they are given to the second tape and then comparing them in reverse order with the y_j, it is easy to see that the problem is in $\mathrm{ST}(2, O(\log N), 2)$, where $N = \Theta(m \cdot \log m)$ is the size of the input. Thus we get:

Proposition 10. *For all functions $r, s : \mathbb{N} \to \mathbb{N}$ with $r(N) \cdot s(N) \in o(N)$,*

$$\mathrm{ST}(2, O(\log N), 2) \ \not\subseteq \ \mathrm{ST}(r, s) = \mathrm{ST}(r, s, 1).$$

Note that several external memory tapes do not help if no head reversals are permitted, that is, $\mathrm{ST}(1, s, t) = \mathrm{ST}(1, s)$ for all $s : \mathbb{N} \to \mathbb{N}$ and $t \geq 1$.

While several external memory tapes enable us to copy large segments of the input tape from one place to another, the segments themselves remain unchanged. In particular, there seems no easy way to "significantly" re-order the input or large parts of it. This leads to the idea that *sorting* should be hard even in the model with several external memory tapes.

Sort
Input: $x_1, \ldots, x_m \in \{0,1\}^n$.
Output: $x_1, \ldots, x_m$ sorted in ascending lexicographical order.

It is not hard to see that the standard *merge-sort* algorithm achieves the following bound:

Proposition 11. Sort *can be solved by an $\big(O(\log m), O(n), 3\big)$-bounded Turing machine.*

The main lower bound result is the following:

Theorem 12 ([12]).

$$\text{Sort} \notin \mathrm{ST}\left(o(\log N),\ O\left(\frac{\sqrt[5]{N}}{\log N}\right),\ O(1) \right).$$

The main ideas of the proof will be outlined in Subsection 5.1 below.

Unfortunately there is still a considerable gap between the lower bound of Theorem 12 and the upper bound of Proposition 11. However, for the important special case of sorting strings of length $O(\log m)$, or equivalently integers in the range $\{0, \ldots, m^{O(1)}\}$, with a little more effort we obtain *tight* bounds. Let us denote the restriction of Sort to instances with $n = 6 \cdot \log m$ by Short-Sort. The factor 6 is needed for technical reasons, any constant above 6 would work as well. Proposition 11 shows that Short-Sort $\in \mathrm{ST}(O(\log m), O(\log m), 3)$, which yields the upper bound of the following theorem. The lower bound is obtained by reducing a restricted version of the sorting problem for "long" strings, which is used in the proof of Theorem 12, to Short-Sort.

Theorem 13 **([12]).** Short-Sort *is in* $\mathrm{ST}(O(\log N), O(\log N), 3)$*, but not in*

$$\mathrm{ST}\left(o(\log N), O\left(\sqrt[6]{N}\right), O(1)\right).$$

By further refining the techniques underlying the proof of Theorem 12, we also obtain lower bounds for the following two related decision problems:

Set-Equality
Instance: $x_1, \ldots, x_m, y_1, \ldots, y_m \in \{0,1\}^n$.
Question: Is $\{x_1, \ldots, x_m\} = \{y_1, \ldots, y_m\}$?

Checksort
Instance: $x_1, \ldots, x_m, y_1, \ldots, y_m \in \{0,1\}^n$.
Question: Is $(y_1, \ldots, y_m)$ the sorted version of $(x_1, \ldots, x_m)$, that is, $\{x_1, \ldots, x_m\} = \{y_1, \ldots, y_m\}$ and $y_1 \leq \ldots \leq y_m$ with respect to the lexicographical order ?

Theorem 14 **([13]).**

$$\text{Set-Equality, Checksort} \ \notin \ \mathrm{ST}\left(o(\log N), O\left(\frac{\sqrt[5]{N}}{\log N}\right), O(1)\right).$$

Since both Set-Equality and Checksort are easily reducible to Sort, similarly as for Short-Sort, we obtain matching upper bounds for the restrictions of the problems to input strings of length $O(\log m)$.

The complement of the restriction of Set-Equality to instances with $n = 6 \cdot \log m$ can be decided by an $(1, O(\log m), 1)$-bounded nondeterministic Turing machine, which just guesses an x_i which is different from all y_j, stores it on an internal memory tape, and compares it to all y_j. Thus we get:

Corollary 15 **([13]).**

$$\mathrm{NST}(1, O(\log N), 1) \ \not\subseteq \ \mathrm{ST}\left(o(\log N), O\left(\frac{\sqrt[5]{N}}{\log N}\right), O(1)\right).$$

On the other hand, the Set-Equality problem can be expressed by the xquery of Figure 6.
We therefore obtain:

Corollary 16. *There is an xquery Q such that*

$$Q\text{-Filtering} \notin \mathrm{ST}\left(o(\log N),\ O\left(\frac{\sqrt[5]{N}}{\log N}\right),\ O(1)\right).$$

```
<result>
  if ( every $x in /instance/set1/string satisfies
         some $y in /instance/set2/string satisfies $x = $y )
     and
     ( every $y in /instance/set2/string satisfies
         some $x in /instance/set1/string satisfies $x = $y )
  then <true/>
  else ()
</result>
```

Fig. 6. An xquery that checks whether two sets are equal

5.1 List machines and the proof of Theorem 12

As pointed out at the beginning of Section 5, arguments that are solely based on communication complexity do not lead to lower bound proofs for the classes $\mathrm{ST}(r,s,t)$ where $t \geq 2$ external memory tapes are available, because the second external memory tape can be used to transfer large parts of the input tape from one place to another.

On the other hand, even with additional external memory tapes, there seems no easy way of significantly *re-order* large parts of the input. In fact, it is well-known that the sorting problem cannot be solved by a *comparison exchange algorithm* that performs significantly less than $m \cdot \log m$ comparisons. Consequently, for sufficiently small $r(N)$ and $s(N)$, even with $t > 1$ external memory tapes, sorting by solely *comparing and moving around the input strings* is impossible. However, this does not lead to a proof of Theorem 12, because the *Turing machines*, on which the $\mathrm{ST}(\ldots)$ classes are based, can perform much more complicated operations than just "compare and move input strings". Indeed, many algorithms that solve certain data stream problems in a surprisingly efficient way are based on much more intricate operations (cf. [18]).

For proving the lower bound of Theorem 12 we introduce a new machine model, so-called *list machines*, which enable us to analyze the flow of information in a Turing machine computation. On the one hand, list machines can only compare and move around input strings as a whole and in this sense are "weaker" than Turing machines. We exploit this weakness in our proof that (appropriately restricted) list machines cannot sort. On the other hand, list machines are non-uniform and have a large number of tape symbols and states. In this sense, they are much stronger than Turing machines.

Here, we only describe list machines informally. For a formal definition and a precise statement of the following Simulation Lemma, we refer the reader to [12].

List machines are similar to Turing machines, with the following important differences:

– They are *non-uniform*; the input consists of m bit strings each of which has length n, for *fixed* m, n.

– They work on *lists* instead of tapes. In particular, this means that a new cell can be inserted between two existing cells.
– Instead of single symbols, list cells contain *strings* over the alphabet

$$A = I \cup \mathsf{states} \cup \{\langle, \rangle\},$$

where $I = \{0,1\}^n$ is the set of potential input strings.
– The transition function only determines the list machine's new state and the head movements and *not what is written into the list cells.*
– If (at least) one head moves, the information w on the current state and the content of *all* list cells that are seen by the machine's read/write heads directly before the transition, is stored on every single list as follows: On those lists whose heads are about to move a step to the left or the right, the information w overwrites the current cell entry. On each of the other lists (i.e., those whose heads do not move in the current step), a *new list cell*, containing the information w, is inserted behind the current head position.

The operation of a list machine is illustrated by Figure 7.

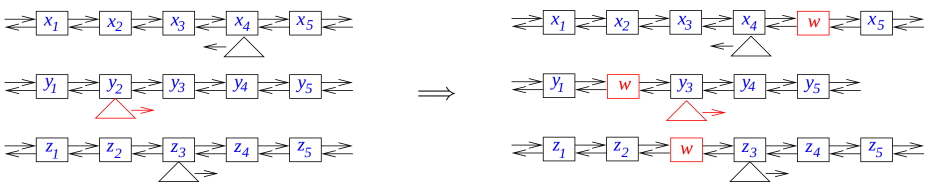

Fig. 7. A transition of a list machine. The example transition is of the form $(q, x_4, y_2, z_3) \to (q', \mathit{stay}, \mathit{right}, \mathit{stay})$. The new string w that is written into the tape cells consists of the current state q and the content of the list cells read before the transition. Formally, we let $w = q\langle x_4\rangle\langle y_2\rangle\langle z_3\rangle$.

The crucial fact is that Turing machines can be simulated by list machines. Informally, the *Simulation Lemma* states that every (r,s,t)-bounded Turing machine can be simulated by a family of list machines with

– $r(N)$ head reversals;
– t lists;
– $2^{O(s(N)+\log N)}$ states.

As usual, N denotes the input size.

The proof of the Simulation Lemma is the technically most difficult part of the proof of Theorem 12.

The next step in the proof of Theorem 12 is to show a lower bound for sorting on list machines. Very roughly, the idea is to analyze the *skeleton* of a list machine computation. The skeleton of a configuration or run of a list machine is obtained by replacing all input strings (of size n) by their indices (of size $\log m$). Intuitively, the skeleton determines the flow of information during a run, but not the outcome of the comparisons. Here we say that two input strings are *compared* if they appear together in a list cell in the run. Now counting arguments, based on the fact that there are not too many skeletons, show that there is are inputs $\bar{v} := (v_1, \ldots, v_m)$ and $\bar{v}' := (v'_1, \ldots, v'_m)$ in I^m and an index $i \in \{1, \ldots, m\}$ such that in the computation of the list machine:

– $\bar{v}$ and $\bar{v}'$ generate the same skeleton.
– $v_i \neq v'_i$, but $v_j = v'_j$ for all $j \neq i$.
– When v_i printed as part of the output, then no head reads a list cell that contains v_i (this information depends only on the skeleton), and thus the machine cannot know if is supposed to print v_i or v'_i. Thus for one of the two inputs, it will print the wrong string.

This shows that the machine cannot sort. Combined with the Simulation Lemma, it yields a proof of Theorem 12.

References

[1] G. Aggarwal, M. Datar, S. Rajagopalan, and M. Ruhl. On the streaming model augmented with a sorting primitive. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, pages 540–549, 2004.

[2] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximationg the frequency moments. *Journal of Computer and System Sciences*, 58:137–147, 1999.

[3] A. Arasu, B. Babcock, T. Green, A. Gupta, and J. Widom. Characterizing memory requirements for queries over continuous data streams. In *Proceedings of the 21st ACM Symposium on Principles of Database Systems*, pages 221–232, 2002.

[4] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of the 21st ACM Symposium on Principles of Database Systems*, pages 1–16, 2002.

[5] Z. Bar-Yossef, M. Fontoura, and V. Josifovski. On the memory requirements of XPath evaluation over XML streams. In *Proceedings of the 23rd ACM Symposium on Principles of Database Systems*, pages 177–188, 2004.

[6] Z. Bar-Yossef, M. Fontoura, and V. Josifovski. Buffering in query evaluation over XML streams. In *Proceedings of the 24th ACM Symposium on Principles of Database Systems*, 2005. To appear.

[7] P. Duris, Z. Galil, and G. Schnitger. Lower bounds on communication complexity. *Information and Computation*, 73:1–22, 1987.

[8] G. Gottlob, C. Koch, and R. Pichler. Efficient algorithms for processing XPath queries. In *Proceedings of the 28th International Conference on Very Large Data Bases*, 2002.

[9] G. Gottlob, C. Koch, and R. Pichler. The complexity of XPath query evaluation. In *Proceedings of the 22nd ACM Symposium on Principles of Database Systems*, pages 179–190, 2003.

[10] G. Graefe. Query evaluation techniques for large databases. *ACM Computing Surveys*, 25(2):73–170, 1993.

[11] M. Grohe, C. Koch, and N. Schweikardt. Tight lower bounds for query processing on streaming and external memory data. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming*, 2005. To appear.

[12] M. Grohe and N. Schweikardt. Lower bounds for sorting with few random accesses to external memory. In *Proceedings of the 24th ACM Symposium on Principles of Database Systems*, 2005. To appear.

[13] M. Grohe and N. Schweikardt. Unpublished manuscript, available from the authors, 2005.

[14] M. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. In *External memory algorithms*, volume 50, pages 107–118. DIMACS Series In Discrete Mathematics And Theoretical Computer S cience, 1999.

[15] E. Kushilevitz and N. Nisan. *Communication Complexity.* Cambridge University Press, 1997.

[16] U. Meyer, P. Sanders, and J.F. Sibeyn, editors. *Algorithms for Memory Hierarchies*, volume 2832 of *Lecture Notes in Computer Science.* Springer-Verlag, 2003.

[17] J.J. Munro and M.S. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, 12:315–323, 1980.

[18] S. Muthukrishnan. Data streams: algorithms and applications. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 413–413, 2003.

[19] A. Neumann and H. Seidl. Locating matches of tree patterns in forests. In V. Chandru and V. Vinay, editors, *Proceedings of the 18th Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 1530 of *Lecture Notes in Computer Science*, pages 134–145, 1998.

[20] Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems.* McGraw-Hill, 2002.

[21] A.A. Razborov. Applications of matrix methods to the theory of lower bounds in computational complexity. *Combinatorica*, 10:81–93, 1990.

[22] L. Segoufin and V. Vianu. Validating streaming XML documents. In *Proceedings of the 21st ACM Symposium on Principles of Database Systems*, pages 53–64, 2002.

[23] Luc Segoufin. Typing and querying XML documents: Some complexity bounds. In *Proceedings of the 22nd ACM Symposium on Principles of Database Systems*, pages 167–178, 2003.

[24] J.F. Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM Computing Surveys*, 33:209–271, 2001.

[25] K. Wagner and G. Wechsung. *Computational Complexity.* VEB Deutscher Verlag der Wissenschaften, 1986.

[26] A. Yao. Some complexity questions related to distributive computing. In *Proceedings of the 11th ACM Symposium on Theory of Computing*, pages 209–213, 1979.